

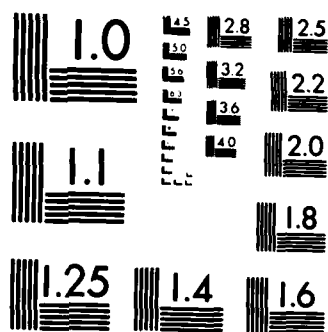
A LOGICAL DESIGN OF A SESSION SERVICES CONTROL LAYER OF 1/1
A DISTRIBUTED NET. (U) NAVAL POSTGRADUATE SCHOOL
MONTEREY CA B A FREQ JUN 84

1/1

F/G 15/5

NL

[illegible]



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A150 486



DTIC
ELECTE
FEB 21 1985

B

THESIS

A LOGICAL DESIGN OF A SESSION SERVICES CONTROL
LAYER OF A DISTRIBUTED NETWORK ARCHITECTURE

FOR SPLICE

by

Barry Albert Frew

June 1984

Thesis Advisor:

Norman F. Schneidewind

Approved for public release; distribution unlimited

DTIC FILE COPY

85 02 01 053

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
	AD-4 156 486	
4. TITLE (and Subtitle) A Logical Design of a Session Services Control Layer of a Distributed Network Architecture <i>FOR SPLICE</i>		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Barry A. Frew		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		12. REPORT DATE June 1984
		13. NUMBER OF PAGES 52
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Logical Design; Session Services; SPLICE; Network: Distributed		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This thesis provides a logical design view of the session services control layer of a distributed network to be used in the SPLICE (Stock Point Logistics Integrated Communication Environment) project. It examines the functional requirements of session services, the data necessary to provide that functionality, and the interfaces required. These areas typically focus on the SPLICE application specifically, but		

DD FORM 1473
JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

1

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

would apply to a generic session services as well.

The recommendations that are offered relate to the SPLICE application more specifically, and address the prospect of placing a fault tolerant capability in session services for SPLICE. Other recommendations are appropriate only to the SPLICE environment.

Approved for public release; distribution unlimited.

A Logical Design of a Session Services Control
Layer of a Distributed Network
Architecture for SPLICE

by

Barry A. Frew
Lieutenant, United States Navy
B.S.A.S., Miami University of Ohio, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

Author:

Barry A. Frew

Approved by:

Norman F. Schneidewind
Thesis Advisor

Daniel R. Dolk
Second Reader

Will. A. Dine
Chairman, Department of Administrative Sciences

Kenneth T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

This thesis provides a logical design view of the session services control layer of a distributed network to be used in the SPLICE (Stock Point Logistics Integrated Communication Environment) project. It examines the functional requirements of session services, the data necessary to provide that functionality, and the interfaces required. These areas typically focus on the SPLICE application specifically, but apply to a generic session services as well.

The recommendations that are offered relate specifically to the SPLICE application and address the prospect of placing a fault tolerant capability in session services for SPLICE. Other recommendations are appropriate only to the SPLICE environment.

TABLE OF CONTENTS

I.	INTRODUCTION	6
II.	NETWORK ARCHITECTURE LEVELS	10
III.	SESSION SERVICES FUNCTIONS	13
IV.	SESSION SERVICES DATA	27
V.	MAPPING OF DATA TO FUNCTIONS	30
VI.	SESSION SERVICES INTERFACES	32
VII.	DESIGN RECOMMENDATIONS	35
VII.	SUMMARY	48
	BIBLIOGRAPHY	49
	LIST OF REFERENCES	50
	INITIAL DISTRIBUTION LIST	51

Accession For

NTS ☒ NAME ☐

☐

☐

☐

☐

Availability Codes

and/or

Special

A-1



I. INTRODUCTION

Navy stock points and inventory control points (ICPs) are currently in a situation which suggests that they are outgrowing their current data processing capabilities. The current hardware suite consists of medium sized Burroughs B-3500/3700/4700/4800 Systems and is becoming saturated by a seemingly endless demand for more support. Each year, at an annual users' conference, the list of "things to do" grows longer. Each stock point is autonomous with respect to how it implements data processing support, as long as it accommodates the Navy Supply Systems Command (NAVSUP) requirements. As a result, various mini and micro computers at these facilities have used locally-provided code instead of the standard code provided by the Fleet Material Support Office (FMSO) for the FMSO-maintained systems. FMSO has a charter to provide system software and application software for the existing data processing systems used at stock points. This system is called the Uniform Automated Data Processing System - Stock Points (UADPS-SP).

Long range plans to overcome the current situation include the purchase of new hardware for the stock points and ICPs. This effort has resulted in a contract with Tandem Computer, Inc. to provide necessary hardware. The design framework in which this hardware is to be utilized is a distributed Local Area Network (LAN) architecture. It has been given the name of Stock Point Logistics Integrated Communications Environment (SPLICE).

SPLICE is designed to augment the existing data processing environment at stock points. UADPS-SP, which runs on the Burroughs Systems mentioned above, is only one of the automated systems at most stock points. There is the Integrated Disbursement and Accounting (IDA) System which is operating on the Interdata 7/32, the Automated Procurement and Data Entry (APADE) System, and the Trident Logistics Data System (Trident LDS). Each has its own data elements, files, programs, transactions, users, reports, and some have additional hardware. To augment them all and not force redesign to existing systems is a difficult task. By the time that SPLICE is ready to be implemented, there will surely be more systems with which to interface.

Many of the new application requirements involve interactive capabilities and telecommunications support. This, in general, is not supportable with the current mix of hardware and software.

The two advertised major objectives of SPLICE are: (1) to allow for CRT display terminals to interact with application logic and to fetch information from a system data base; and, (2) to provide a standard interface for each of the sixty-two supported supply sites. Notice that these goals are stated in data processing rather than functional terms.

The implementation, in general, is envisioned as having mini-computers used as front-ends to the existing hardware. The interfaces with existing systems will be controlled by

the LAN. The Burroughs systems would be able to provide larger file processing functions and report generation functions in a background mode. Each LAN is planned to be "standardized," and have the capability to communicate with other LANs via the Defense Data Network (DDN), which is managed by the Defense Communications Agency.

The functional objects which are contained in the SPLICE network have been identified at a high level of abstraction (Schneidewind and Dolk, Nov 1983), and major functions have been identified for those objects. It is the purpose of this paper to decompose the object called Session Services into lower levels of abstraction, and to clarify the functions at a level which should feed the design of the actual module.

A session can be described as all of the activity which takes place among two or more processes for the duration of a single task. The Session Services module is the object which functions as the liaison between the user and required functional modules. When the user specifies a required task, the session services object will initiate and control the required functional modules to accomplish the requested task, and return the response and the control to the user. This control mechanism is a complex one, primarily due to the following constraints:

- o A user process may have multiple sessions active at any time.
- o A functional module can be active in multiple sessions at a time.

- o Two or more functional modules can be active in multiple sessions at any time.
- o Message exchange between pairs of functional modules can be nested.

The multi-tasking requirements above are in addition to message exchanges between local and remote sites, and tasks may be either deferrable or interactive.

This complex processing environment requires that a module exist to control this activity. This role has been delegated to the Session Services Module.

To more fully comprehend the role of Session Services it may be helpful to examine the layers of control found in a network architecture, of which Session Services is a member.

II. NETWORK ARCHITECTURE LEVELS

A fundamental goal of the SPLICE network architecture is to permit system designs that support geographic distribution of workstations which reflect the natural partitioning (Bachman, 1978) of the Navy activities involved. All application programs must be independent from the physical topology of the nodes where they reside. The Session Services Layer supports this independence. Workstation programs are written to request session establishments among them using only logical addressing names (mailboxes) which are independent from physical topology. These requests are all sent to Session Services. The session services module employs two principal mechanisms (Bachman, 1978). The first mechanism is that it links processes into temporary cooperative relationships by locating the desired partner process via the data dictionary/directory system and activates that process in a workstation after insuring that the partner is ready. An approach for increasing the speed and reducing the overhead by cutting hand shaking procedures to the bone is explained in (Schneidewind and Dolk 1983). The second mechanism employed is the exchange of data and status and control information over established sessions. This synchronizes cooperating processes, the databases they modify and the journal entries of exchanged messages in support of data integrity requirements. Security, resource management, and system administration also have manifestations at the

session service level, but have not been included as major Session Services mechanisms.

Network architectures may vary with regard to implementation structures but, in general, the functions provided are the same. The session services layer is usually one of several layers of control identified within an architecture. Using a logical architecture model closely related to most commercially available architectures is a useful way of examining the entire control structure.

A layered approach to network architecture from the International Standards Organization (ISO) defines seven distinct layers of protocol. (Deitel, 1983)

Physical Layer - This layer handles the mechanical and electrical details of the actual physical transmission of bit sequences over communication lines.

Data Link Layer - This layer controls the manipulation of data packets. It handles the addressing of outgoing packets and the decoding of addresses on incoming packets. It detects and possibly corrects errors that occur in the Physical Layer.

Network Layer - This layer controls the switching and routing of messages between stations in the network.

Transport Layer - This layer provides for transfer of messages between end users. The users need not be concerned with the manner in which reliable and cost effective data transfers are achieved.

Session Layer - This layer provides the means for cooperating processes to organize and synchronize their dialog and manage their data exchange.

Presentation Layer - This layer resolves differences in formats among the various computers, terminals, databases, and languages used in a network.

Application Layer - This layer is the highest layer and

provides services directly to users. It deals with data exactly as it is generated by and delivered to user processes. This level contains the user-specified functions and program controls.

The topology, media, and access procedures for the eventual SPLICE system will not be addressed, as they are implementation features. Instead, the emphasis will be placed on design considerations which impact the system requirements.

The next section will use hierarchical decomposition to aid in the definition of the Session Services control functions.

III. SESSION SERVICES FUNCTIONS

Each of the major functions will be decomposed into their logical pieces and described in more detail.

The first step is to describe the functions that are within the scope of the design. No attempt is made to constrain this step by conforming to current organizational structure or data processing systems in use. No attempt is made to determine the likelihood of automation. This functional decomposition is not based on organizational attributes, nor does the decomposition address control issues at any level. Naming the business functions as accurately and consistently as possible helps users relate to the formalized function descriptions during the requirements review. At each aggregate level of abstraction, an attempt was made to maintain a similar level of detail throughout that level. That is, the scope, size, magnitude, and relative importance of each module should be approximately equal at each level. During the process of function definition, the next lower level of functionality is described in an effort to validate the level being developed, and to insure the ability to make accurate tradeoff decisions concerning cohesion and coupling issues later. The function description consists of a unique identifying number, the logical function name, a description of the function, and its required input and output. When validating functions with respect to system requirements, it is useful to have a hierarchical chart of the functions

available to cross-reference the function with its relative position among the functions as a whole. The data flow diagram is also a helpful tool to relate the flow of data through the different functions. Particularly confusing ideas may also be charted in an effort to clarify the particular environment or requirement.

The following diagram shows the major logical components of the session services module:

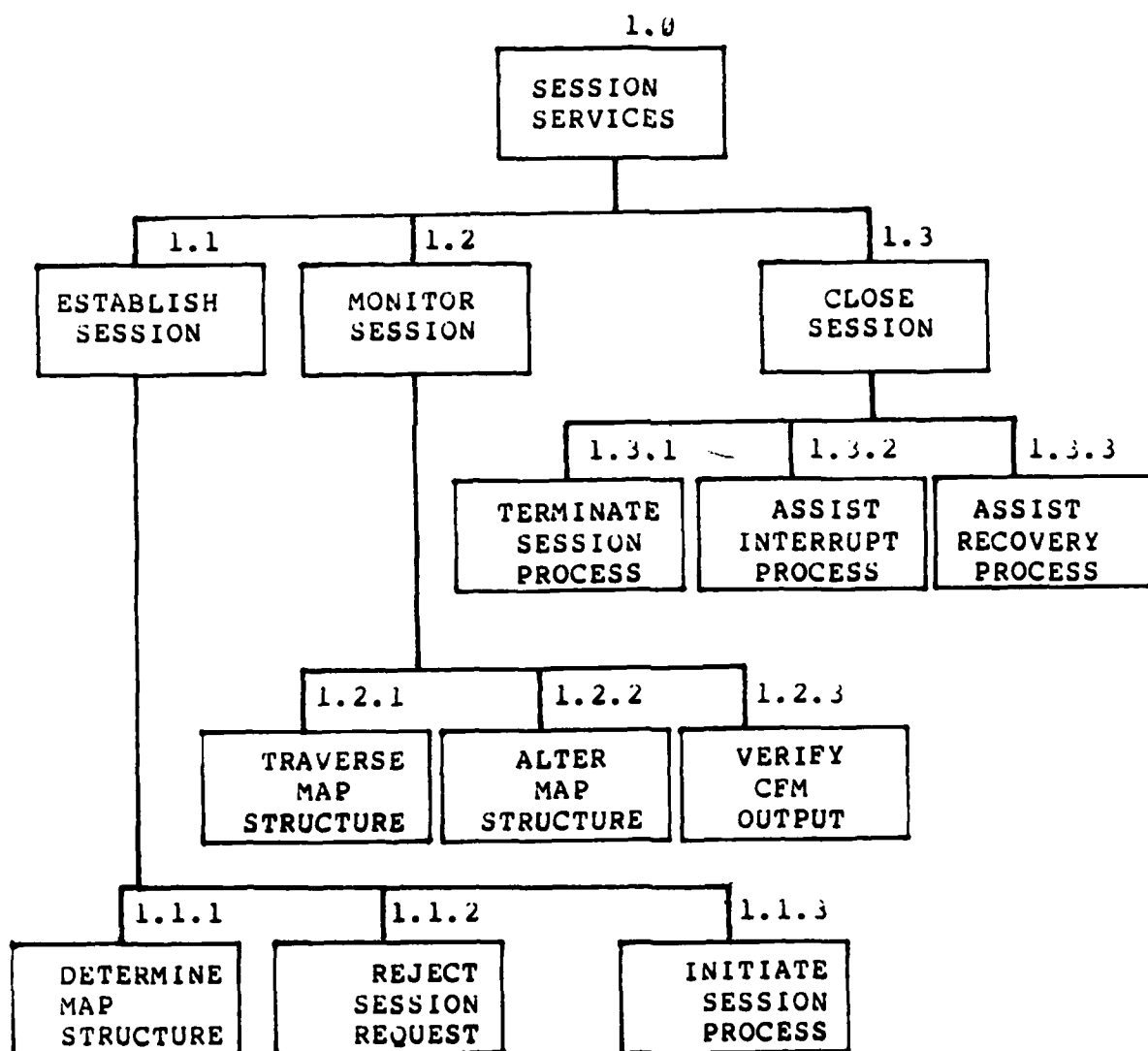
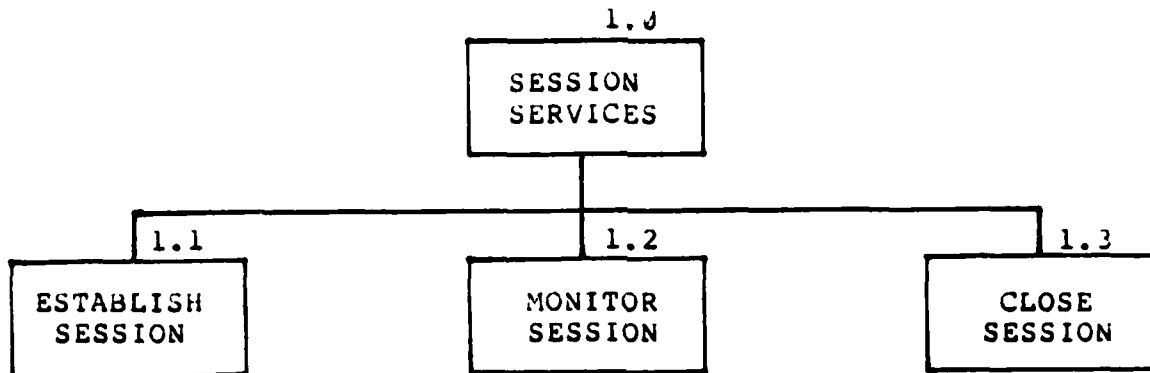


Table 1

The following descriptions are for high level functions. Because each high level function is subsequently broken into lower level functions, the descriptions will only include a narrative describing the function performed. The lower level functions will then be described in more detail and inputs and outputs will be identified.



NAME: SESSION SERVICES
IDENTIFIER: 1.0

Establishes, monitors, and closes all sessions on the system. It initiates and controls the required controlling functional modules to accomplish the requested task(s), and returns a response and/or control to the user via the TM module upon completion. This module is capable of controlling multiple sessions per user.

NAME: ESTABLISH SESSION
IDENTIFIER: 1.1

Translates a service request from the Terminal Management Module (TM) to its required mapping of functional modules to satisfy the task named in the service request. The function also activates the initial Controlling Functional Module (CFM) and sets session status to indicate an active session, or rejects the session request based on security, or user authority criteria.

NAME: MONITOR SESSION
IDENTIFIER: 1.2

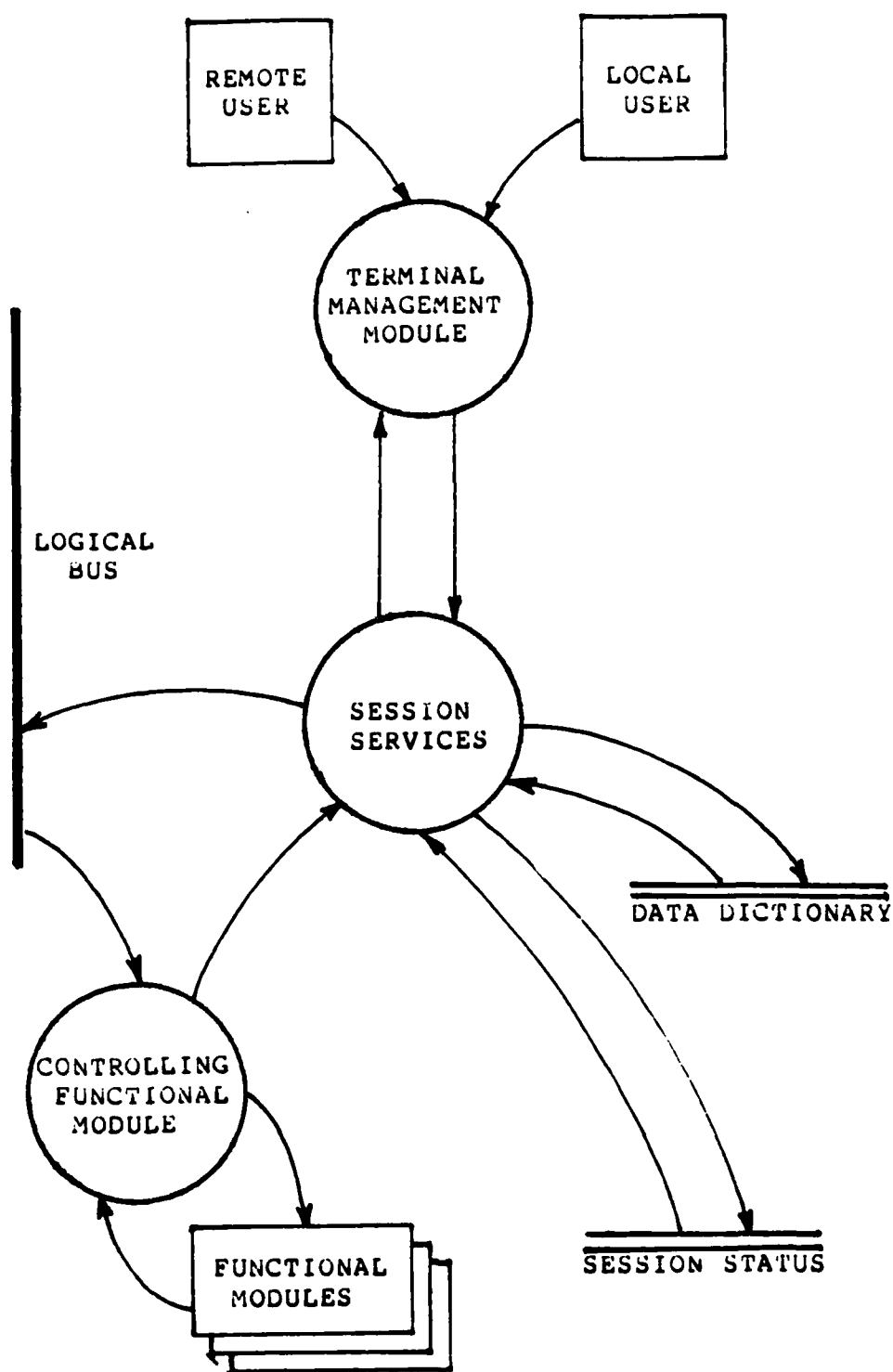
Maintains communication with, and passes control to each CFM in accordance with the session map. This function identifies additional data required for the user to complete a given task based on the map of functional modules. This function also makes any changes to the functional module map, and will verify output from a CFM for those tasks which contain fault tolerant backup modules.

NAME: CLOSE SESSION
IDENTIFIER: 1.3

Updates session status to show result of Session Complete Message, aids the system in maintaining consistency in the session and/or system status during interrupt processes, and aids the Recovery Module by providing accurate session status information.

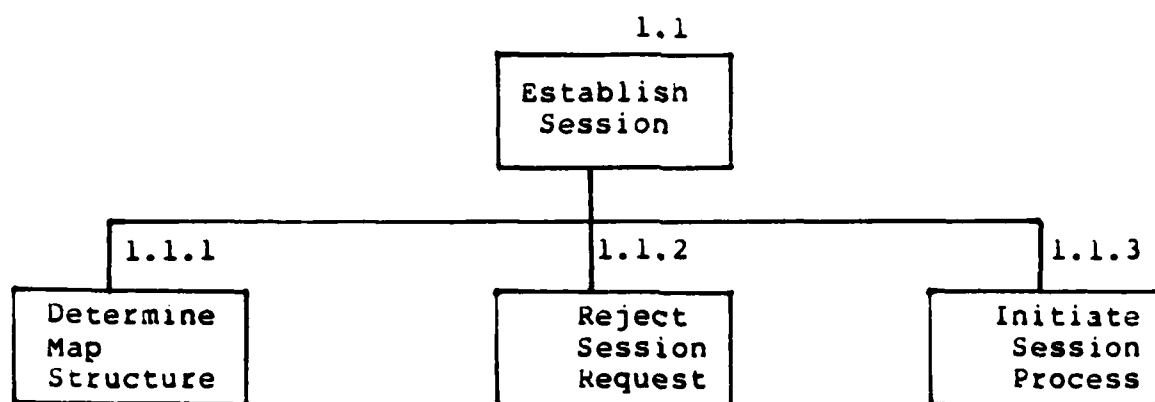
The following data flow diagram is an example of the system activity that occurs at the highest level. The data flow diagram is most useful to the reader as a verification that the system is consistent and logical. It is sometimes easier to view the system in this chart form.

HIGH LEVEL
DATA FLOW DIAGRAM



The next section describes the lowest level functions. At this level, the modules will have suggested logical inputs and outputs identified for later mapping to a low level data flow diagram. Inputs and outputs have been associated with their respective sources (origins) and destinations (target). The sources and destinations are SPLICE modules and CFMs. When either the source or the destination is a sub-module within the Session Services Module it will be identified by the unique identifier of the sub-module. To later complete the logical design for the specific SPLICE application, the input and output data must be quantified. The expected level of activity will help to size the system and provide necessary input to design decisions concerning data base structure as well as module determination.

The decomposition that follows is for the Establish Session function.



NAME: DETERMINE MAP STRUCTURE

IDENTIFICATION: 1.1.1

DESCRIPTION: This function determines the correct map structure based on the task request and user or system provided parameters. The map structure is a definition of the processes within a task and their relationships to one another. Task requests may be directed to a specific workstation mailbox by means of a unique logical name. This information is used to enter the data dictionary/directory system and retrieve the associated mapping structure for the required functional modules.

SOURCE/DESTINATION

INPUT: Session Request	Terminal Management Module
OUTPUT: Task Request	Data Dictionary Functional Module

NAME: REJECT SESSION REQUEST

IDENTIFICATION: 1.1.2

DESCRIPTION: This function receives the exception status from the DD/DS and sends an appropriate response to the originator via the TM.

SOURCE/DESTINATION

INPUT: Task Rejection	Data Dictionary Functional Module
OUTPUT: Session Rejection	Terminal Management Module

NAME: INITIATE SESSION PROCESS

IDENTIFICATION: 1.1.3

DESCRIPTION: This function assigns a unique identifier to the session, initiates a session-data entry which includes the map structure, a timestamp of initiation, current controlling functional module, etc., flags the controlling functional module as "active," and passes control via a message to the local or remote CFM. (The National Communication Module will relay this message

over the DDN if it is destined for a remote CFM.

SOURCE/DESTINATION

INPUT: Map Structure

Data Dictionary Functional
Module

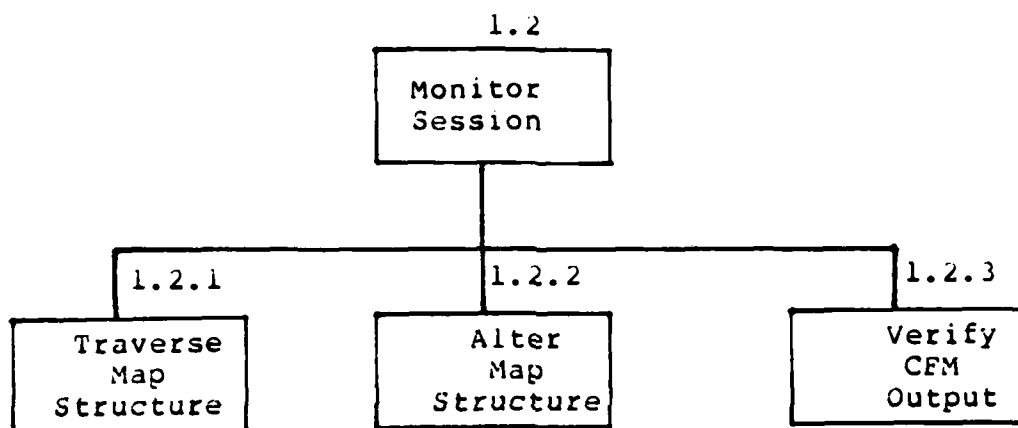
```

OUTPUT: Controlling
        Functional
        Module Name

```

Local or remote CHM

The decomposition that follows is for the Monitor Session function.



NAME: TRAVERSE MAP STRUCTURE

IDENTIFICATION: 1.2.1

DESCRIPTION: This function interprets completion messages, output text location(s), and/or parameters from active Controlling Functional Modules (CFMs). It checks the message against the active map and identifies the next CFM to be activated. It removes the activation pointer from the current CFM and places it on the next CFM to be initiated by sending a message to the CFM. If the map has been completely traversed, then a completion message and the location of output text and/or parameters are sent to the Terminal Management Module (TM). (Graphical view of the possible process state transitions are in Table 1 on page 21.) This module also receives abnormal termination messages from CFMs which have fault tolerant backup modules and replaces

the module producing the error with a replacement module.

SOURCE/DESTINATION

INPUT:	CFM Completion Message	Controlling Functional Module (CFM)
	Output Text Location	CFM
	Control Parameter	CFM
OUTPUT:	Session Completion Message	Terminate Session Process (1.3.1)

PROCESS STATE TRANSITIONS

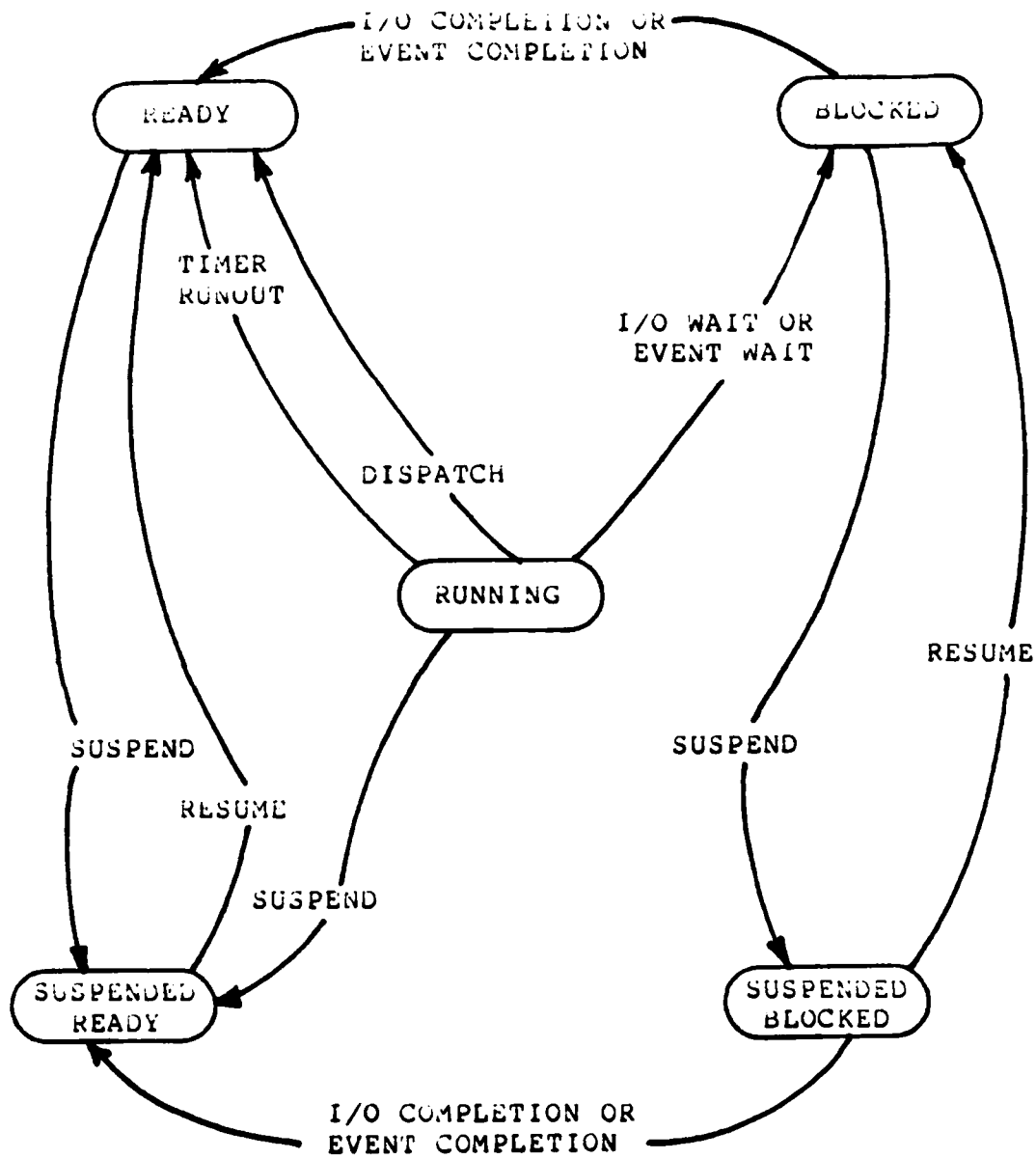


TABLE 2

NAME: ALTER MAP STRUCTURE

IDENTIFICATION: 1.2.2

DESCRIPTION: This function is responsible for any alteration made to the functional map associated with a task. Data can be received from the active controlling functional module, the fault tolerant output verification module (1.2.3) or the user. The data may consist of calls to modules that are outside of the normal map structure, requests for additional information, or data describing interference situations caused by access of shared resources.

SOURCE/DESTINATION

INPUT: FM Call Validate Request	CFM
User Message	Terminal Management
OUTPUT: Map Alteration Message	Traverse Map
	Structure (1.2.1)
FM Call Validation Response	CFM

NAME: VERIFY CFM OUTPUT

IDENTIFIER: 1.2.3

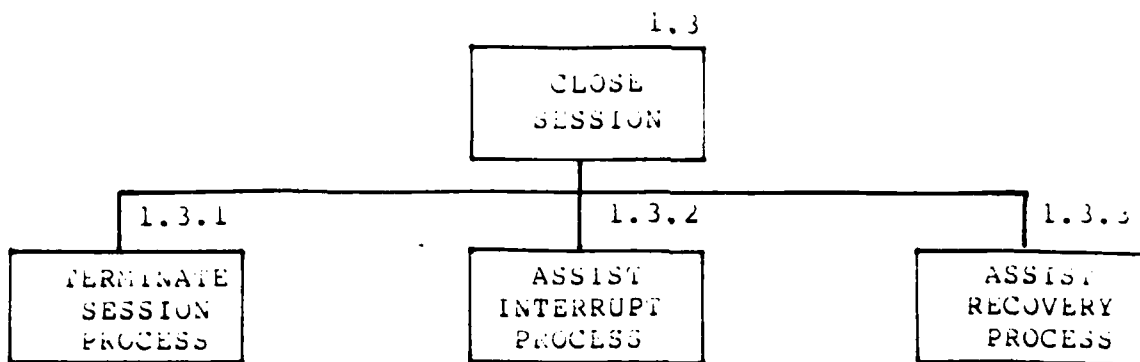
DESCRIPTION: This function verifies output from Controlling Functional Modules prior to control being transferred from the CFM. It compares the output data that has resulted from the CFM against a prescribed valid output standard that has been developed for the module. The output will either match, meaning that control can be passed because the module arrived at valid data, or the output will not be considered valid, meaning the module has failed. When the module has failed to provide valid data, the name of the logical fault tolerant module replacement is sent to Alter Map Structure and the replacement module will be executed.

SOURCE/DESTINATION

INPUT: CFM Output	CFM
OUTPUT: Replacement Module Name	Alter Map Structure
	(1.2.2)
Message to User	Terminal Management
Error Notification	CFM

The decomposition that follows is for the Close Session

function.



NAME: TERMINATE SESSION PROCESS

IDENTIFIER: 1.3.1

DESCRIPTION: This function is responsible for completing the session-status data entry for a completed map structure, and signifying the end of a session. The "active" flags will be removed from the last active CFM, timestamped for termination, and an appropriate message sent to the TM module indicating reason for termination and location of any existing output.

SOURCE/DESTINATION

INPUT: Map Complete Message	Traverse Map Structure (1.2.1)
OUTPUT: Output Location	TM module
Termination Message	TM module

NAME: ASSIST INTERRUPT PROCESS

IDENTIFIER: 1.3.2

DESCRIPTION: This function will resolve interrupts originating when the active controlling functional module requires additional data concerning user desires or system requirements.

SOURCE/DESTINATION

INPUT: CFM Request	CFM
User Request	TM module

OUTPUT: User Request
CFM Request

TM module
CFM

NAME: ASSIST RECOVERY PROCESS

IDENTIFICATION: 1.3.3

DESCRIPTION: This function aids the Recovery Module (RM) by providing current status of all sessions upon request. The restart and/or recovery is controlled entirely by the RM module.

SOURCE/DESTINATION

INPUT: Status Request

RM module

OUTPUT: Status Response

RM module

The following chart shows a summary of possible commands for the Session Services module:

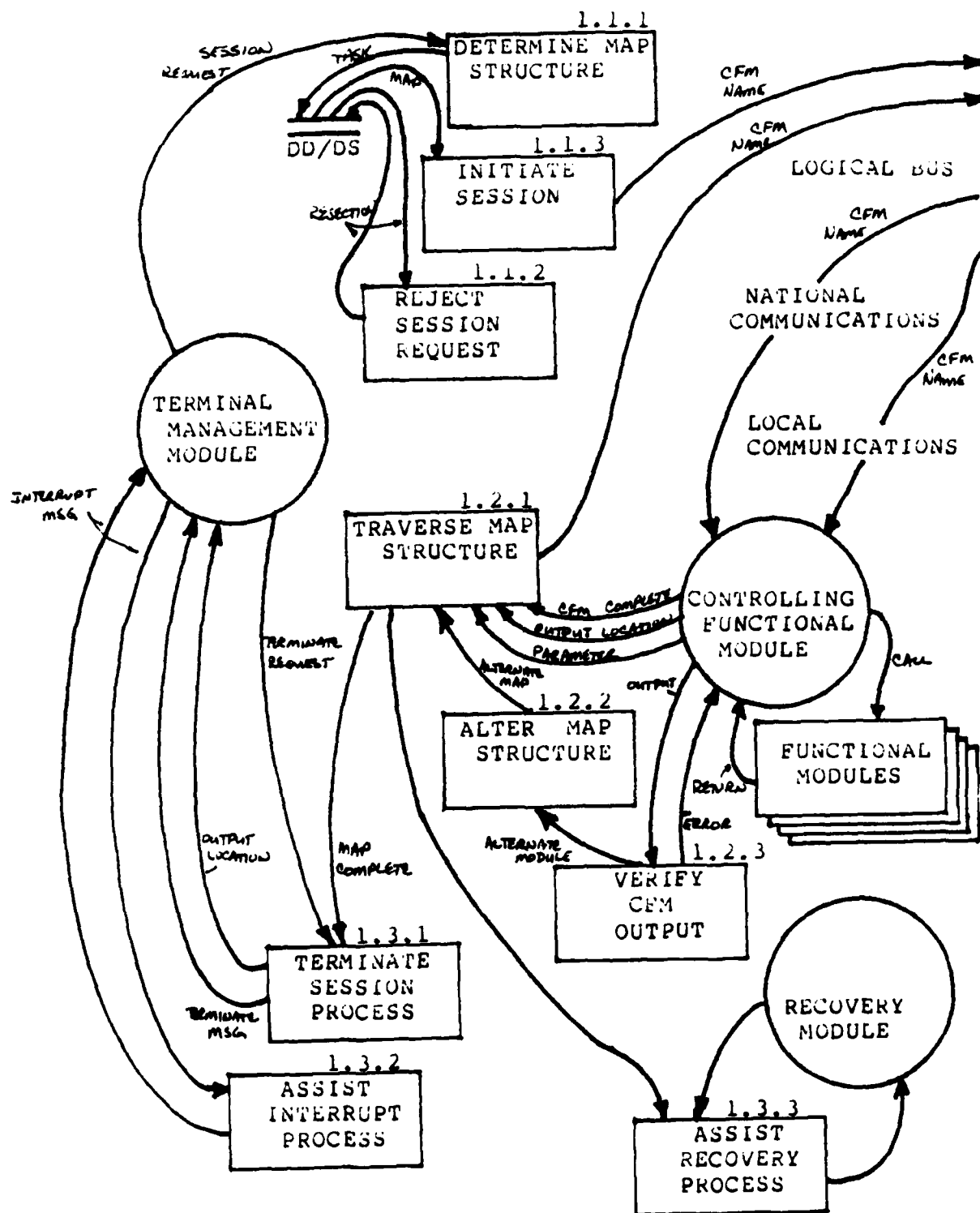
INITIATE ACCEPT TERMINATE	}	SESSION
SEND RECEIVE	}	SESSION-DATA-UNIT
CANCEL TERMINATE	}	SESSION-QUARANTINE-UNIT
INTERRUPT TERMINATE	}	SESSION-INTERACTION-UNIT
CHECKPOINT RECOVER	}	PROCESS RECOVERY UNIT
COMMIT RECOVER	}	PROCESS-COMMITMENT-UNIT

Table 3

The following data flow chart is an example of the system activity that occurs at the lower level. It is useful both as a different, pictorial view of the interaction, and

as a tool when mapping the data to the functions.

LOW LEVEL
DATA FLOW DIAGRAM



IV. SESSION SERVICES DATA

The next step is to define the data used within the scope of the system. At first, this is done by isolating the view to data only, ignoring any relationship to functions. Because many misconceptions about definitions of terms may exist and may detract from the document's readability, several significant terms associated with the interpretations of data used in this document are defined below:

DATA CLASS - a collection of data used to describe an entity which is easily describable, readily comprehensible and meaningful (within the system boundaries), past the point of just being a collection of data elements. Usually they can be uniquely identified, and have unique data elements associated with them. Basically, two types are distinguishable: objects and concepts.

OBJECT - a particular occurrence of a data class which exists, and is capable of being sensed.

CONCEPT - an effort or action in the real world. An example of some typical concepts might make the distinction clearer. A "bank account" may well be a concept data class. It is uniquely defined by an account number, has meaning within the banking industry, yet you cannot touch it. It exists merely as an agreement between the customer and the bank. It can be represented or referred to with listings of account status, or inquiry, but cannot be seen or sensed.

INDICATIVE KEY - the attribute which enables an object or concept within a data class to be uniquely identified. An indicative key points to one and only one occurrence of a data class.

XREF KEY - an attribute used to show a relationship of one data class to another data class. The physical implementation may appear as a link record or pointer.

Each data class is described in a form consisting of a data description, key elements, and relationships to other

data classes. Selection of the data class name should be consistent with the usage within the system being modeled. That is, the users of the system should not be forced to change their normal vocabulary. The identification of the keys are helpful later in the development of the optimum data structure.

The definition of raw data for the system is completed when this step is completed. Further refinements and subsequent information may require any of the steps to be re-evaluated and altered.

DATA CLASS DESCRIPTIONS

NAME: SESSION - Describes the time frame that represents a user requesting a task from the SPLICE system and receiving a response to the request. The beginning is marked by the Session Services Module after it has received clearance for a particular user for a particular functional module. The end is marked by the Session Services Module when an error has forced an abend status or the functional map has been completed, and control is returned to the user via the Terminal Management Module.

ELEMENTS:

- Session Number (Indicative key)
- Timestamp
- Functional Map
- Controlling Functional Modules
- Current Controlling Functional Modules
- User Identifier (XREF to User)
- Terminal Identifier (XREF to Mailbox)
- Service Request Code (XREF to Task)
- Status Code (Inactive, suspend, resume, I/O wait etc.)

NAME: TASK - Describes each job that the SPLICE system allows the user to request from any work station in the system. It may relate to a single Functional Module, called a Controlling Functional Module, or it may be a mapping of several modules containing a Controlling functional Module and one or more submodules.

ELEMENTS:

- Service Request Code (Indicative key)
- User Authority Required
- Logical Name of Module
- Logical Location
- Physical Location
- Parameter Specification

NAME: MAILBOX - A method of uniquely identifying repositories of data from deferred tasks. It also identifies the nature of the service requested for those users with multiple mailboxes.

ELEMENTS:

- Mailbox Number (Indicative key)
- Command Location
- Logical Address
- Physical Address
- Hardware Configuration
- Clearance Level
- Constraints

NAME: USER - A person or group of persons who are authorized access to the SPLICE system.

ELEMENTS:

- Password (Indicative key)
- Log-on
- User Name
- Parent Command
- Authorization Level
- Terminal Restrictions (XREF to Mailbox)

V. MAPPING OF DATA TO FUNCTIONS

The logical design, based on function and data was presented in the previous section. Future work in this area will involve mapping the data to the functions, that is, we will determine which functions use which data.

The data are categorized with respect to the role they play for that function as well. Data can be a trigger that causes the function to occur; it can be input from another function, or outside source; it can be input from the data repository; it can be an output which triggers another function; it can be output to a data repository; or it can be output to another function or outside destination.

After data have been mapped to the individual functions, a determination is made of what transformations occur in the system is made. A transformation is an action that changes the input data and creates or helps to create an output. Using input/output data defined in the function description, values for the quantities of each particular input and output can be used to determine the numbers of occurrences of these transactions. The quantities may address data class abstractions; therefore, the data dictionary may have to be used to cross reference data elements that are at the particular level identified in the transformations.

This information can be used to graph the number of occurrences of transforms and relate this information to the required data relationships. Using this graph, the

transformations that most frequently occur within the system are identified. This graph can then be used to decide which logical data relationships to implement.

VI. SESSION SERVICES INTERFACES

The Session Services Module has no interface external to the SPLICE system. The user and other systems only access the session services module via other SPLICE modules. There are major interfaces which must be defined within the SPLICE system. The following interfaces are addressed on a logical basis without identifying the actual implementation details:

SESSION SERVICES TO TERMINAL MANAGEMENT MODULE

Rather than communicate directly with user terminals, the functionality is separated between Session Services and Terminal Management. The end user always communicates with the Terminal Management Module and the Terminal Management Module always communicates with Session Services or the user. The TM sends session requests from a particular mailbox or interactive user to Session Services. The TM receives the completion message, the output location, the rejection message, and the information request message from session services. The TM will provide all necessary message editing, screen management and virtual terminal operating functions.

SESSION SERVICES TO RECOVERY MODULE

Session Services is a support module to the Recovery Module. When the Recovery Module requires data concerning the current or recoverable state of all or any session, the request will be addressed to Session Services. Session Services will access the required data and forward them to the Recovery Module. No recovery logic is contained in the Session Services Module.

SESSION SERVICES TO DATA DICTIONARY/DIRECTORY

When Session Services requires data concerning user authority, task security level, task mapping structures, data structures, or functional module relationships, it will request that data from the Data Dictionary/Directory.

SESSION SERVICES TO NATIONAL COMMUNICATION MODULE

During a session, the control is passed to the Controlling Functional Modules via the logical bus mechanism. If the logical name refers to a remote module then it is picked up by the National Communication Module (NC) and initiated for transmission on the DDN. The NC

Module will convert LAN protocol to Defense Data Network protocol and vice versa, enabling session services to rely only on the local LAN protocol.

SESSION SERVICES TO LOCAL COMMUNICATION MODULE

During a session, control is passed to the Controlling Functional Module via the logical bus mechanism. If the logical name relates to a module contained in the LAN, then it is picked up by the Local Communication Module and initiated.

SESSION SERVICES TO SESSION STATE DATA

Session Services is the only module able to access session state data. The Session Services Module will maintain state information in its own file. These data should be structured so that no other module may access them, and more critically, alter them.

SESSION SERVICES TO FUNCTIONAL MODULES

There is no interface between Session Services and functional modules, other than a Controlling Functional Module. All called functional modules interface solely with the CFM.

In general the interfaces do not require any additional information to be added to the LAN message format presented in (Schneidewind, 1982). The data that are contained in the LAN message format include:

- MESSAGE TYPE
- DATE AND TIME
- DESTINATION ADDRESS
 - DESTINATION LOGICAL ADDRESS
 - DESTINATION PHYSICAL ADDRESS
- SOURCE ADDRESS
 - SOURCE LOGICAL ADDRESS
 - SOURCE PHYSICAL ADDRESS
- NUMBER OF FRAGMENTS
- MESSAGE NUMBER
- FRAGMENT NUMBER
- ACKNOWLEDGEMENT FRAGMENT NUMBER
- DATA LENGTH
- SERVICE REQUEST CODE
- DATA
- ERROR CHECK
- FLAGS

All fields are fixed length except for the field called
DATA.

VII. DESIGN RECOMMENDATIONS

Our major design recommendation concerns the implementation of fault tolerant techniques, which is covered in some detail because of the general lack of awareness concerning fault tolerant methods. The other recommendations address possible problem areas related to the nature of the application environment.

A. FAULT TOLERANCE

The notion of incorporating a means for tolerating faults in order to improve computer system reliability is well established. Because of the usual multiple meanings of data processing terms, we will attempt to define several terms. Fault tolerance is a term describing a system capable of coping with faults without a requirement for manual intervention. Fault prevention, a similar concept, avoids potential faults by detecting and eliminating them prior to operating the system. Once the system is operational, fault tolerance, if it exists, begins. The SPLICE session services module should employ fault tolerance to improve reliability.

For a system to be fault tolerant, it must be able to: detect errors, assess and confine the damage, and repair or recover from the error without forcing manual intervention. Detection methods are most abundant. Assessing, confining, and recovering methods are not well defined nor readily available.

Unmastered complexity at any level increases the possibility of a fault occurring. It is highly unlikely that

the SPLICE system will ever be fault free. A major part of the complexity in SPLICE will be implemented in software. Therefore, software developers should seek not to design "perfect, error free" software, but rather attempt to provide reliable fault tolerant techniques for SPLICE in all modules.

Any definition of the reliability of a system must involve distinguishing between acceptable and unacceptable behavior of the system. There must be some method to distinguish between unsatisfactory behavior which is a consequence of user misunderstanding and unacceptable behavior due to deficiencies in the system itself. The system specification should provide that mechanism.

Ideally, a specification should be consistent, authoritative and so complete that the behavior of the system is defined for all possible input and output sequences. In each circumstance where this is not so, acceptable behavior cannot be distinguished from unacceptable behavior. Regardless of the specification, only two concepts are essential to understand the causes of failure: (1) an event (state) which should not have occurred and (2) a condition (state) which should not have arisen. Internally, these are usually referred to as erroneous transitions and erroneous states. If either of these internal situations exists, then it follows that the system has had a component or design failure.

"An erroneous transition of a system is an internal state transition to which subsequent failure could be attributed. Specifically, there must exist a possible sequence of interactions which would, in the absence of corrective action from the system, lead to a system failure attributable to the erroneous transition."

"An erroneous state of a system is an internal state which could lead to a failure by a sequence of valid transitions. Specifically, there must exist a possible sequence of interactions which would, in the absence of corrective action by the system and in the absence of erroneous transitions lead from the erroneous state to a system failure." (Anderson and Lee, 1983)

Design faults are unpredictable and unexpected. Unanticipated errors are the result. Experience from previous, identical or similar faults is usually not helpful in resolving the design fault.

If these ideas are acceptable, then it is not difficult to determine why a major portion of the effort in fault tolerance has been aimed at component failure to date. To illustrate the point, make this trivial comparison of an easy component failure and an easy design failure:

COMPONENT - A diode fails, causing an open circuit. It was predictable since diodes only last so many hours in certain environments. The result of a continually open circuit is predictable and can be anticipated. It is repaired in the same manner as the last diode failure.

DESIGN - A logic circuit fails to produce the desired results. The desired results are to exchange the values of variable A_i and A_j without using an

auxiliary variable.

The algorithm used:

$A_i := A_j$

$A_j := A_i$

An obvious design failure causes the value A_i to be lost.

To correct the algorithm the following adjustment may be made:

$A_i := A_i - A_j$

$A_j := A_j + A_i$

$A_i := A_j - A_i$

The problem did not go away; it just changed and became more difficult to find. Now, it works for most cases; however, this algorithm will fail when $i = j$. The resultant error may not always be the same. The repair of the error may require different procedures for each error, therefore, the experience of past identical or similar failures may not be useful.

Techniques such as top down development, structured programming, step-wise refinement, and information hiding all embrace the principle of divide and (nope to) conquer.

Some of the problems of software-oriented fault tolerance are unique to the errors found in software design. There have been two methods suggested for providing software with fault tolerant characteristics which address those unique qualities which make software fault tolerance so difficult. The methods are called: "Recovery block scheme", and "N-version programming". Both operate under the

assumption that, despite the use of fault prevention techniques, any complex software system will always contain residual faults when placed in service.

The recovery block scheme was designed as a method to provide fault tolerant capabilities to sequential programs. The application module is called the primary module and is defined as a non-redundant software module which has been designed and implemented to satisfy the authorizing specification.

The first stage of the process is to detect when an error arises during the execution of the "primary module." Since the primary module has been debugged and tested as much as practicable, the module may run several times without any error conditions. There are advantages and disadvantages to placing an error detector within the code itself. The error detection module should be separate and executed immediately after the execution of the "primary module," before any data are transferred and used in a subsequent module. This "acceptance test" consists of a sequence of statements which will raise an exception if the state of the system is not acceptable. The primary module will have failed if any exception is raised.

The second stage is to assess the damage and repair or recover. This function will require a recovery point mechanism to prevent rerun of entire modules or groups of modules. Therefore, at the beginning of each primary module, there will be a sequence of statements, or some method of

indicating the status of the machine environment. Various methods are in wide use today and may be applied. When the error occurs and is detected by the "acceptance test," there must be an available alternate module to process. If we were to recover and repeat the primary module under the same conditions, we would expect the error to recur and we would have an endless cycle of execution. The alternate modules are logically capable of producing the required output as defined by the "acceptance test." They may be less efficient with respect to memory and/or speed, however, or be designed in a less complicated fashion. Regardless of their weaknesses, their strength is that they have less probability of design error. Perhaps several alternate modules are developed, each less complicated than the previous. This alternate module is, in fact, built-in redundancy and reflects the degree of reliability required of the system. The alternate modules are invoked only if an error occurs, thus minimizing the overhead. Software monitors can be used to record activation occurrences of these alternate modules. Run time overhead is incurred for each recovery point established and for each call to the acceptance test module. The recovery block logic appears as:

```
ESTABLISH RECOVERY POINT
EXECUTE PRIMARY MODULE
APPLY ACCEPTANCE TEST TO RESULT
INVOKE ALTERNATE MODULE IF FAILURE EXISTS
APPLY ACCEPTANCE TEST TO RESULT
INVOKE ALTERNATE MODULE IF FAILURE EXISTS
```

The usual syntax associated with the scheme is:

```

ENSURE <acceptance test criteria>
BY <primary module name>
ELSE BY <alternate module #1>
ELSE BY <alternate module #2>
ELSE ERROR

```

These blocks can be nested and do not impose constraints on the programming style or the methodology being used. They are compatible with structured programming techniques and high level languages.

In a sort application, the fault tolerant recovery block might appear as:

```

ENSURE  $A[j+1] \geq A[j]$  for  $j=1,2,\dots,n-1$ 
BY Sort A using quicksort
ELSE BY Sort A using shell sort
ELSE BY Sort A using insertion sort
ELSE ERROR

```

Where quicksort is a time-efficient, slick, compactly-coded module; shell sort is less efficient, but easier to visualize and less prone to design error; and insertion sort is a simple brute-force type of sort routine.

Fail soft approaches also use this technique by having each alternate module provide some reduced service or a subset of the required output. The following is an example of a disk-to-memory transfer function which is being gracefully degraded:

```

ENSURE consistency of disk transfer queue
BY enter request in optimal queue position
ELSE BY enter request at end of each queue
ELSE BY send warning message 'request ignored'
ELSE ERROR

```

While the above example may cause problems for the program requesting the transfer, the rest of the system can proceed without disruption.

Control flow of the recovery process can be directly supported by microcode, or by the use of special purpose instructions which are tailored to suit recovery block schemes. The Anderson and Kerr report describes an experimental architecture which provides this support in a recovery cache mechanism.

So far, it has been assumed that exceptions occurring during the execution of a recovery block program will result in backward error recovery and a transfer of control to the next alternate module.

There is, however, nothing to prevent a forward error recovery technique within a module of a recovery block. An application of this concept might be used in the detection of an underflow occurrence. An error handler could insert the lowest value number used in the machine and continue to process; then flag the result with a note to the user that the substitution was made. In this case, it is more efficient than finding a recovery point and running an alternate module. Other situations exist which favor forward error recovery as well.

The recovery block scheme is conceptually simple. The implementation may not be as simple. Alternate modules require extra coding and require more complicated module interfaces. This added complexity has not been quantifiable to date. This is true because of the lack of total testing results and the lack of use in multiple application areas. Because alternate modules are independent of one another,

the implementation of multiple alternate modules is not much more difficult than just one alternate module, if previous modules exist. This could be very expensive if all modules were developed from scratch; therefore, a good source of alternate modules is found in the prior versions of a module. For example, as updated replacement modules (with optimized features, or refined functions) are ready to be implemented, the previous module may be used as the first alternate.

The acceptance test procedure adds a dimension of complexity that is not present in non-fault tolerant systems, although, for every set of alternate and primary modules, this acceptance test need only be designed and implemented once. In the SPLICE application, this module should not be unacceptably large or complex. It should be noted that SPLICE will use a Tandem Computer system which is designed to provide redundant hardware and software modules for all major functions.

The second method for implementing software fault toleration is conceptually simpler. The N-Version approach to fault tolerance has N versions, where N is greater than 1, of independently designed code which satisfies a single specification. All the versions are executed and all the results are compared. The correct (majority) response is sent to its destination while the erroneous responses, if they exist, are ignored.

The implementation of the scheme requires a driver program to invoke each version, collect each response, compare the results, and determine the correct response. Each version operates atomically and uses the same input space, so the driver must synchronize the execution of versions. The originators of this approach, (Chen and Avizienis, 1981), use a handshake method with "wait" and "send" primitives to insure that only one version at a time is running. This scheme prevents realizing the advantages of pipelining, or parallel processing. It also requires a timeout detection capability to prevent infinite loops. An advantage of the N-version technique is that damage assessment would not have to be done at all, and error recovery is done by simply ignoring the erroneous answer. An obvious disadvantage on the other hand is the overhead of atomically executing each version with the same input space, and executing a voting check to obtain a single result.

There are few, if any, theoretical reasons for not adopting fault tolerance techniques. It seems that the real factors preventing widespread acceptance are:

- o Education: Lack of courses, books, and quantifiable evidence of improved reliability, make it a difficult idea to sell to managers.
- o Psychological: Adopting these schemes is, in some way, admitting the existence of design faults. Therefore, the designers and programmers do not have motivation to persuade the managers.
- o Cost: It is perceived, by project managers, as an additional cost without additional functionality. Without the necessary motivation, they are unable or unwilling to justify the development or runtime overhead involved.

Since a great deal of money will be spent on SPLICE software (development and maintenance), producing fault tolerant software may be the most advantageous approach from a systems life cycle cost perspective. Again, we note that hardware fault tolerance is provided by the SPLICE Tandem system. However, this feature is of little help, if there are errors in applications software.

B. Jeopardizing Coupling and Cohesion Principles

A designer should not attempt to excessively enlarge functional modules in an effort to reduce overhead. This must be a carefully considered option. If multiple functions are combined or extra data are passed to prevent another call, then the modular characteristics begin to deteriorate. This may result in a very expensive and time consuming maintenance portion of the life cycle. Controlling functional modules should be capable of coordinating an entire transaction. For a single controlling module to coordinate an entire transaction, it is required that it contain all logic necessary to perform that transaction. System designs which dictate that coded modules be as cohesive as practical and that coupling interfaces be kept simple, are usually easier to maintain. It is far better, at least in the design phase, to split out functions in the widest pattern. More breadth in design eases the task of maintaining coupling and cohesion principles, while not adversely impacting the implementation. If, for example, after sizing the transactions, it appears that control

cannot be achieved efficiently, it is possible to systematically combine the modules with the least negative impact and the most benefit.

C. Use of ADA Program Unit Specifications

The use of ADA as a program design language would serve to enhance the SPLICE software design process. The use of ADA program unit specifications during design can be implemented in either ADA or Pascal easily. Additionally, this systematic approach, will allow this design to interface more easily with future DOD supported projects in the logistics environment. The ADA Program Support Environment (APSE) could have a long learning period associated with it. To begin using it now would improve FMSO's long term position with respect to the Navy's support of DOD policy and, at the same time, provide a good environment in which to learn it with relatively low risk. APSE would provide configuration control of the functional allocation of each program unit.

D. Naming of Modules

The naming of unique modules will be difficult to achieve because of the widespread adoption of local unique programs at each of the stock points. Many stock points have personalized the reorder process, the inventory process, and many management reports. In some stock points, local unique programs form the basis of the MIS used, instead of the FMSO provided UADPS-SP. Local programs may have the same names as modules at another stock point; they may also have the

same name and function as a module from UADPS-SP. Because most stock points believe they have unique requirements or because the wait to get requested UADPS-SP enhancements is excessively long, they have created a number of local-unique programs that enhance UADPS-SP or in some cases replace it. Transaction Item Reporting (TIR) notices from the stock point to the ICPs for ICP-controlled items is largely done with stock point local-unique programs.

This is a critical problem to solve before the design of Session Services or any other module goes too far. If the local unique programs cannot be standardized for all stock points, or cannot be placed in a global library with unique names and parameters identified, the entire level of control and ability to share data and processes envisioned by SPLICE proponents is at risk.

VII. SUMMARY

The Session Services Module should contain functions as described in section III with the the data described in section IV as a support structure. The interfaces described in section VI must be kept in mind during the detail design phase of the project. The major design recommendation offered in this paper is the fault tolerant concept. Design recommendations to prevent large modules which hinder the coupling and conesion characteristics of modules and the use of ADA program unit specifications are mentioned because the SPLICE environment may be well served by their use. The recommendation to standardize the local unique program names is discussed in an effort to prevent the serious situation that would result if this aspect of the SPLICE project were ignored. The stock points which will be served by this system have a long history of addressing local requirements with locally designed and developed software whicn interfaces with FMSO-provided software. All of the stock point requirements must be examined carefully and a decision that is mutually agreeable between designers and users must be made concerning what parts are to be standard and unmodified, and which parts are to be changed by local adaptations. The eventual use or misuse of the system could be determined by this decision.

BIBLIOGRAPHY

- "Constructing User Interface Based on Logical Input Devices," Computer, pp. 62-68, November, 1982
- "Heuristic Models of Task Assignment Scheduling in Distributed Systems," Computer, pp. 54-56, June, 1982
- Davies, L.W., Computer Networks and Their Protocols, John Wiley and Sons, 1979
- Jacobsen, Tom, et. al., "Virtual Terminal Protocols Transport Service and Session Control", Computer Communication Review, ACM, Vol. 10, No. 1 and 7, pp 24-40, January/April 1980.
- Loomis, Mary E.S., Data Communications, Prentice-Hall, 1983.
- Pooch, Udo W., Greene, William H., and Moss, Gary G., Telecommunications and Networking, Little, Brown and Company, 1983.
- Shoch, J.F., "Internetwork Naming, Addressing and Routing", Proceedings of COMPCON Fall 78, IEEE Computer Society, pp.72-79.
- Sultzter, Jerome H., "On the Naming and Binding of Network Destinations Local Computer Networks," Ravasco, Piercarlo, et. al. (eds.), North-Holland Publishing Co., pp.311-317, 1982.
- Tanenbaum, Andrew S., Computer Networks, Prentice Hall, 1981.

LIST OF REFERENCES

Anderson, T. and Kerr, R., "Recovery Blocks in Action: A System Supporting High Reliability," Proceedings of 2nd International Conference on Software Engineering, San Francisco (CA), pp. 447-457, October 1976.

Anderson, T. and Lee, P.A., Fault Tolerance Principles and Practice, Prentice Hall, 1981.

Avizienis, Algirdas, "Fault Tolerant Computing: An Overview", Computer, pp. 5-8, Jan/Feb 1971.

Bachman, Charles and Canepa, Mike, "The Session Control Layer of an Open System Interconnection", Proceedings, Computer Communications Networks, COMPCON, pp 150-156, September 1978.

Deitel, H.M., An Introduction to Operating Systems, Addison-Wesley, 1983.

Schneidewind, Norman F., "Functional Design of a Local Area Network for the Stock Point Logistics Integrated Communications Environment", Naval Postgraduate School, Monterey, December 1982.

Schneidewind, Norman F., "Functional Approach to the Design of a Local Network: A Naval Logistics System Example", Digest of Papers, Spring COMPCON 83 pp.197-202.

Schneidewind, Norman F., and Dolk, Daniel R., "A Distributed Operating System Design and Dictionary/Directory for the Stock Point Logistics Integrated Communications Environment", Naval Postgraduate School, Monterey, November 1983.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Prof. Norman F. Schneidewind Code 54Ss Administrative Sciences Department Naval Postgraduate School Monterey, Ca 93943	1
2. Prof. Dan Dolk Code 54Dk Administrative Sciences Department Naval Postgraduate School Monterey, Ca 93943	1
3. Library Code 0142 Naval Postgraduate School Monterey, Ca 93943	1
4. Lt. Barry A. Frew Code 54Fw Administrative Sciences Department Naval Postgraduate School Monterey, Ca 93943	4
5. Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
6. Curricula Officer Code 37 Computer Technology Naval Postgraduate School Monterey, CA 93943	1
7. LCDR Ron Nichols Code 94L Fleet Material Support Office Mechanicsburg, PA 17055	1
8. LCDR Ted Case Code 94L Fleet Material Support Office Mechanicsburg, PA 17055	1
9. Commander Dana Fuller Code 0415A Commander, Naval Supply Systems Command Washington, D.C. 20376	1

END

FILMED

3-85

DTIC